Evolutionary Art Using the Fly Algorithm

Abbood, Zainab; Amlal, Othman; Vidal, Franck

Applications of Evolutionary Computation

DOI:
10.1007/978-3-319-55849-3_30

Published: 01/01/2017

Peer reviewed version

Z. Ali Abbood, O. Amlal, and Franck P. Vidal: Evolutionary Art using the Fly Algorithm. In *20th European Conference on the Applications of Evolutionary Computation*. Volume 10199 of Lecture Notes in Computer Science (LNCS), pages 455-470, Amsterdam, The Netherlands, April 2017. Springer.

DOI: 10.1007/978-3-319-55849-3_30

Keywords: Digital mosaic, Evolutionary art, Fly algorithm, Parisian evolution, cooperative co-evolution.

```
@inproceedings{Abbood2017EvoIASP,
  author = {Zainab Ali Abbood and Othman Amlal and F. P. Vidal},
  title = {Evolutionary Art using the {Fly} Algorithm},
  booktitle = {Applications of Evolutionary Computation},
  year = 2017,
  series = {Lecture Notes in Computer Science},
  volume = 10199,
  pages = {455-470},
  month = Apr,
  address = {Amsterdam, The Netherlands},
  annotation = {Apr~19--21, 2017},
  publisher = {Springer},
  ISBN = {978-3-319-55848-6},
  doi = {10.1007/978-3-319-55849-3_30},
  abstract = {This study is about Evolutionary art such as digital mosaics.
    The most common techniques to generate a digital mosaic effect heavily rely on
    Centroidal Voronoi diagrams. Our method generates artistic images as
    an optimisation problem without the introduction of any a priori knowledge or
    constraint other than the input image. We adapt a cooperative co-evolution strategy
    based on the Parisian evolution approach, the Fly algorithm, to produce artistic visual
    effects from an input image (e.g.~a photograph). The primary usage of the Fly algorithm is
    in computer vision, especially stereo-vision in robotics. It has also been used in image
    reconstruction for tomography. Until now the individuals correspond to simplistic primitives:
    Infinitely small 3-D points. In this paper, the individuals have a much more complex
    representation and represent tiles in a mosaic. They have their own position, size, colour,
    and rotation angle. We take advantage of graphics processing units (GPUs) to generate
    the images using the modern OpenGL Shading Language. Different types of tiles are
    implemented, some with transparency, to generate different visual effects, such as digital
    mosaic and spray paint. A user study has been conducted to evaluate some of our results.
    We also compare results with those obtained with GIMP, an open-source software for
    image manipulation.},
  keywords = {Digital mosaic, Evolutionary art, Fly algorithm, Parisian evolution,
    cooperative co-evolution},
  pdf = {pdf/Vidal2010EvoIASP.pdf},
}
```

Evolutionary Art using the Fly Algorithm

F. P. Vidal [1], and P.-F. Villard [2]
[1] School of Computer Science, Bangor University, LL57 1UT, United Kingdom
[2] LORIA, University of Lorraine, France

Abstract

This study is about Evolutionary art such as digital mosaics. The most common techniques to generate a digital mosaic effect heavily rely on Centroidal Voronoi diagrams. Our method generates artistic images as an optimisation problem without the introduction of any *a priori* knowledge or constraint other than the input image. We adapt a cooperative co-evolution strategy based on the Parisian evolution approach, the Fly algorithm, to produce artistic visual effects from an input image (e.g. a photograph). The primary usage of the Fly algorithm is in computer vision, especially stereo-vision in robotics. It has also been used in image reconstruction for tomography. Until now the individuals correspond to simplistic primitives: Infinitely small 3-D points. In this paper, the individuals have a much more complex representation and represent tiles in a mosaic. They have their own position, size, colour, and rotation angle. We take advantage of graphics processing units (GPUs) to generate the images using the modern OpenGL Shading Language. Different types of tiles are implemented, some with transparency, to generate different visual effects, such as digital mosaic and spray paint. A user study has been conducted to evaluate some of our results. We also compare results with those obtained with GIMP, an open-source software for image manipulation.

Keywords: Digital mosaic, Evolutionary art, Fly algorithm, Parisian evolution, cooperative co-evolution.

1 Introduction

The boundaries between artists and computer scientists may become thinner as the technology becomes more and more ubiquitous. A relatively new field of computer graphics (CG) is called non-photorealistic rendering (NPR). One of the main goals of NPR is to produce "digital art" that can benefit the artistic community as well as the scientific community, e.g. in scientific and medical visualisation [13]. Rendering algorithms have been proposed to simulate multiple forms of traditional art, e.g. digital watercolours [5], line art drawing [18], expressive painting [3], and Celtic art [14]. This paper focuses on the most ancient of classical art forms, mosaics, but also includes other types such as spray paint.

A digital mosaic tries to provide artistic touches to a source image by covering it by tens, hundreds, or thousands of small coloured square tiles in a way that resembles ancient mosaics or stained-glass windows. The main goal is to generate a discrete coloured image that still gives the same impression as the real image. To design a piece of mosaic, an artist needs to precisely decompose the original image into tiles with different size, colour, and orientation. The artist requires a large area where to fit the tiles together like a jigsaw forming a special image (it is not unusual to have mosaics over several square metres) [6, 1].

In image processing and computer vision, the approach consists of building an algorithm that produces an image with mosaic effects automatically or with as little user intervention as possible. The produced mosaic image should replicate the features of the real image [7]. One of the difficulties in digital mosaic generation is that the same original image may be visualised into various mosaics. Therefore, choosing an appropriate tile data set (including tile number, position, size, colour and rotation of every tile) will impact onto the final mosaic image.

Mosaic images can be categorised into four types: i) crystallization mosaic, ii) ancient mosaic, iii) photo mosaic, and iv) puzzle image mosaic. The first two types of mosaics are traditional. The mosaic is the

reconstruction of a real image using small square tiles. The last two types are obtained by aggregating multiple small images to approximate the real image.

In this article, we revisit digital mosaic-like image generation. Our method is also suitable for other effects, such as spray paint. The image generation is considered as an optimisation problem (image reconstruction) and we propose to solve it using artificial evolution (AE), in a particular cooperative co-evolution (CoCo) strategy. Our method relies on the Fly algorithm [19]. To validate our results, a user study has been conducted. It is used to ascertain which version of our algorithm produces the most visually appealing results. We also demonstrate the ability of the algorithm to preserve edges and compared some of our results with similar ones produced with GNU Image Manipulation Program (GIMP) (`http://www.gimp.org/`), an open-source software for image editing.

Section 2 discusses previous work. It primarily focuses on digital mosaic, which is the problem the closest to the one considered in this paper. The following section is a general overview of the "Parisian evolution" strategy and of the Fly algorithm. Section 4 describes our approach. It explains how the Fly algorithm can be adapted from robotic applications and medical tomography reconstruction into an evolutionary art generator. The penultimate section presents our results. Several images have been generated with different versions of the algorithm. We conducted an experiment with 25 participants to judge some these results. Concluding remarks are given in the last section.

2 Previous work

To our knowledge Haeberli is the first researcher who worked on digital mosaic [9]. He created attractive images using an ordered collection of brush strokes to create mosaic and paint effects. He generated images by regulating the colour, shape, size, and orientation of individual brush strokes. To control the mosaic effect his method heavily relies on Voronoi diagrams. One of the main limitations of his algorithm at the time is that it took several hours to produce a satisfactory image. However, much less time should be required with today's "massively parallel processors". This is actually the approach followed by Hoff and his colleagues to overcome the limitation mentioned above. They presented an implementation to compute discrete Voronoi diagrams on graphics processing units (GPUs) [11]. The method starts with a set of random points representing various sites in the image. They are used as the basis to create polygonal meshes that can be rendered in OpenGL to create the Voronoi diagrams. Their approach relies on a metrics based on the Euclidean distance for each site, which computes the distance from any point to that site. Each site has a unique colour.

Hausner improved Hoff's method to use regular and square tiles only. The aim is to create images that have an effect similar to actual mosaics [10]. Each tile may have a different size, colour, and orientation based on the image considered. This approach relies on Centroidal Voronoi (CV) diagrams, which usually order points in regular hexagonal grids. Instead of using the Euclidean distance as a metrics, the Manhattan distance is preferred to place the tiles in different orientation following the edges of the original image.

Lai *et al.* [17] extended Hausner's work by trying to place mosaic tiles on a surface. The tiles are located over a mesh model that is created using a CV diagram and the Manhattan distance. The size of tiles is regular, i.e all the tiles have the same shape (rectangle) and size. The orientation of tiles depends on a vector field, which is interpolated over the surface based on control vectors. The algorithm is sensitive to sharp creases, open boundaries, and boundaries between regions of different colours, which may affect the orientation of tiles.

Lu *et al.* presented a hybrid method that combines Centroidal Voronoi Tessellation (CVT) and Monte Carlo with minimisation (MCM). CVT places the tiles on a mesh surface. Because of local minima, MCM is applied to optimise the result of CVT on a global basis, which improves the final results [22].

In 2015, Hu and his colleagues presented an algorithm for the reconstruction of digital surface mosaics based on irregularly shaped tiles [12]. They use a hybrid optimisation paradigm, which includes continuous configuration optimisation and discrete combinatorial optimisation. In the continuous configuration optimisation scheme, the tiles are adjusted using iterative relaxation. The aim is to adapt their position, orientation, and scale to fit onto approximated Voronoi regions. The aims of the discrete combinatorial optimisation are to reduce the amount of overlapping tiles and to increase the surface coverage.

Nguyen *et al.* [25] produced digital images using an evolutionary algorithm (EA) based on multi-dimensional archive of phenotypic elites (MAP-Elites). Their aim was to demonstrate that deep neural

networks (DNNs) can be easily fooled. Their implementation evolves a population to produce a tremendous diversity of images with a strong chance that DNN can classify the objects correctly.

Another approach is the use of Evolutionary art [4]. In this context, an EA somehow generates images. Artificial evolution is used to modify the images (e.g. by mutation and recombination). Evolutionary art is often an interactive task where the user/artist plays the role of a selection operator. Our work follows the Evolutionary art paradigm. We provide a method without the need of any user interaction, without constraints such as the requirement to generate a Voronoi diagram, and limit the amount of *a priori* knowledge to the input image.

3 Fly algorithm paradigm

We saw in the previous section various approaches to translate an input image into a digital mosaic. The problem can be defined as follows:

> *Given a rectangular region I^2 in the plane $\mathbb{R}^2$, a dataset of N tiles, a set of constraints, and a vector field $\phi(x, y)$ defined on that region, find N sites $P_i(x_i, y_i)$ in I^2 to place the N tiles, one at each site P_i, such that all tiles are disjoint, the area they cover is maximized and the constraints are verified as much as possible.*

In this context, image generation can be studied as a special case of the *set cover problem*, which is NP-complete [15]. It can be solved as an optimisation problem [10, 1]: Find the best set of tiles to generate a rectangular region to approximate a coloured image. Each tile has a colour that represents the specific part of the image it covers. For more realistic reconstruction, the tiles can rotate at a given angle $\phi(x, y)$ following the direction field for that region and may have sightly different sizes. If there are N tiles to place, as each tile has 9 parameters (3-D position, 3 colour components, width, height, and rotation angle), the search space has $9 \times N$ dimensions.

In addition to being a difficult optimisation problem to solve, the digital mosaic generation is also related to topics in computer graphics and visualisation. In particular we saw in Section 2 that most of the mosaic synthesis methods are based on Centroidal Voronoi. In this paper, we propose to solve such a problem without the use of any Voronoi diagram. Instead we rely on an unsupervised EA based on cooperative co-evolution principles.

The approach we follow is called "Parisian evolution". In classical EAs the best individual of the population corresponds to the solution of the optimisation problem, i.e a global optimum. In the Parisian approach all the individuals of the population (or at least a subset of the population) is the solution: Each individual only encodes a part of the solution, and they have to collaborate to build the final solution. Fig. 1 illustrates the mechanics of the Parisian evolution. A Parisian EA usually contains all the usual components of an EA (i.e. genetic operators such as selection, mutation, and recombination), plus the additional components as follows:

- **2 levels of fitness**
  - *Global fitness* computed on the whole population,
  - *Local fitness* computed on each individual to assess their own contribution to the global solution.

  The global fitness may be the sum (or a complex combination) of the local fitnesses but not necessarily. The local fitness of an individual may be defined as its marginal contribution to the global fitness.
- **A diversity mechanism** to avoid individuals gathering in only a few areas of the search space.

The Parisian approach shares many similarities with the Cooperative Co-Evolution Algorithm (CCEA). Similar internal evolutionary engines are considered in classical EA, CCEA and Parisian evolution. The difference between CCEA and Parisian evolution resides in the population's semantics. CCEA divides a big problem into sub-problems (groups of individuals) and solves them separately toward the big problem [24]. There is no interaction/breeding between individuals of the different sub-populations, only with individuals of the same sub-population. However, Parisian EAs solve a whole problem as a big component. All population's individuals cooperate together to drive the whole population toward attractive areas of the search space.

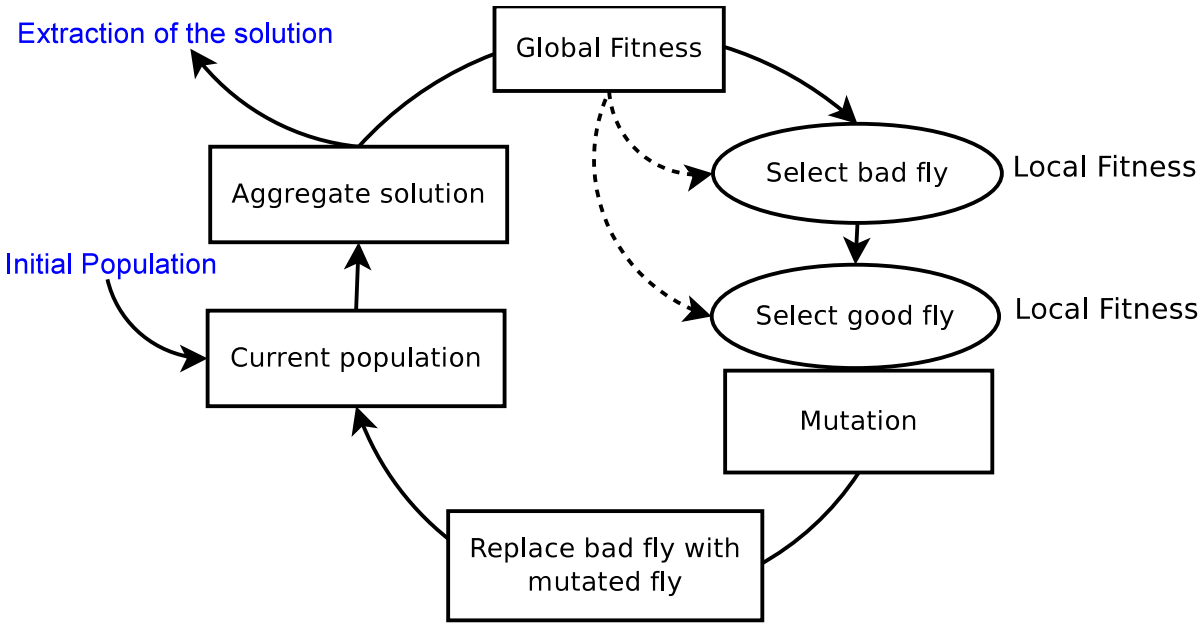


Figure 1: Steady-state Parisian evolution algorithm.

One good example of Parisian EA is the Fly algorithm. Here an individual is a 3-D point. The position of each fly is optimised using the repetitive application of genetic operators. The global and local fitness functions are the key elements of the algorithm. The final set of points is the solution of the optimisation problem.

It has first been applied in computer vision, particularly stereovision, and robotics [19], where flies are projected to gather on the surface of objects. It has been successfully applied to autonomous robots for obstacle avoidance [2, 20] and to self localisation and mapping (SLAM) [21]. Another computer vision application is related to hand gesture recognition [16]. Computer vision is not the only field where the Fly algorithm has been applied. It has been tried in tomography reconstruction in nuclear medicine, where the concentration of flies approximate a radioactive concentration within the human body [26]. Our approach is closely related to this work.

4 Evolutionary image reconstruction

The individuals correspond to extremely simple primitives: The flies. To date, the Fly algorithm has been used to find 3-D positions only. In this paper we propose to give flies a finite size so that they now correspond to rectangular tiles. This is because hand-crafted mosaics tend to use such a shape and also because paint brush strokes could be represented using patterned rectangles. Each fly is a vector of 9 elements (see Fig. 2):

Position is a 3D point with coordinates (x, y, z), which are randomly generated between 0 and $width - 1$, 0 and $height - 1$, and 1 and -1 respectively (with $width$ and $height$ the number of pixels in the image along the x- and y-axes). An example of an image generated by an initial population is shown in Fig. 3.

Colour has three components (r, g, b) (for red, green and blue), which are randomly generated between 0 and 1. This is to ensure diversity at the start of the optimisation. Tile colours are evolved rather than assigned deterministically. It leads to better results in term of sharpness when tiles are located at edges between different regions of the image (see the difference between Figures 6a and 6b, when tile colours are evolved, and Figures 6c and 6d, when the tile colours are not evolved).

Rotation Angle is randomly generated between 0 and 360.

Scaling factor has two components (w, h), which control the size of the tile along its horizontal and vertical axes.

Local fitness measures its marginal contribution toward the global solution.

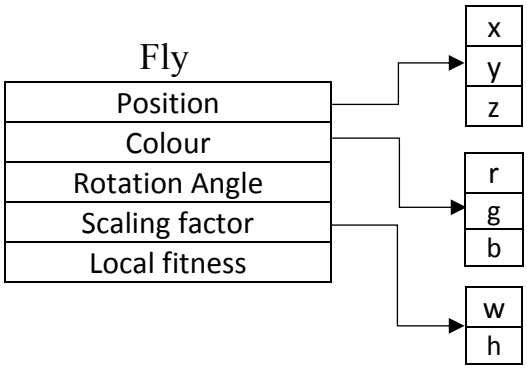


Figure 2: Structure of the fly data.

The initial scaling factors are set to make sure the tiles could cover the totality of the image [10]. If there are N individuals, the scaling factors are:

$$d = \sqrt{(width \times height)/N} \tag{1}$$

Due to the randomness in the initial tile positions, tiles overlap. It creates holes in Fig. 3. They progressively disappear during the evolution process, which aims to optimise the 9 parameters (position, colour, scale, and rotation) of all the N individuals. To achieve this, the algorithm minimises the global fitness function. To assess how good the population is, we compare the input image (ref) with the image generated using the tiles corresponding to the population (pop). We use the sum of absolute error (SAE) (also known as Manhattan distance) to quantify the error between ref and the computed image pop:

$$\text{SAE}(pop, ref) = \sum_i \sum_j |ref(i,j) - pop(i,j)| \tag{2}$$

To improve the population's performance, we need a large proportion of good individuals. The performance of a single fly is evaluated using the local fitness function, which is used during the selection process. In our context, the local fitness is called "marginal fitness", F_m) (see Eq. 3). It measures the impact of the selected fly on the global performance of the population. To measure how good or bad the contribution of

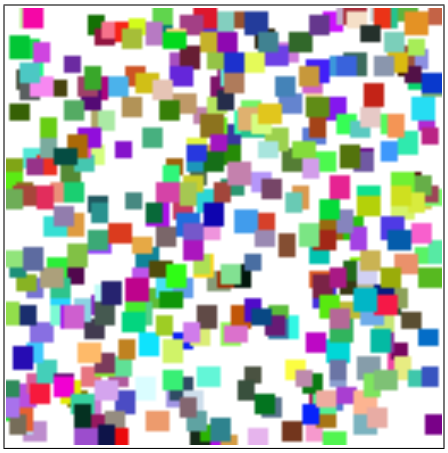


Figure 3: Random initial population.

Fly i is, we use the SAE metrics with the leave-one-out cross-validation method:

$$F_m(i) = \text{SAE}(pop - \{i\}, ref) - \text{SAE}(pop, ref)) \tag{3}$$

with $pop - \{i\}$ the image computed with all individuals but Fly i. The numerical value of $F_m(i)$ can be easily interpreted by looking at its sign:

- If the error is greater with Fly i than without, $\text{sgn}(F_m(i)) < 0$, then Fly i damages the performance of the population.
- If the error is smaller with Fly i than without, $\text{sgn}(F_m(i)) > 0$, then Fly i has a positive impact on the performance of the population.
- If the error is the same, $\text{sgn}(F_m(i)) = 0$, then Fly i is not beneficial nor detrimental. It may happen when Fly i is covering similar flies.

We use this principle in our *Threshold selection* operator [26]. To find a fly to kill, pick a random number i between 0 and $N - 1$. If $F_m(i) \leq 0$, then Fly i can be killed, if not pick another random number i until $F_m(i) \leq 0$. To find a fly to reproduce, find one whose fitness $F_m(i)$ is strictly positive. During the evolution process, the number of flies whose fitness is negative or null will decrease. There will be more and more good flies; and fewer and fewer bad flies: It gives a good stopping criteria to the algorithm as the selection operator will struggle to find bad flies to kill.

Computing the marginal fitness for the problem considered here is time consuming on a central processing unit (CPU). Therefore, all the computations to generate images are performed on a GPU using the OpenGL Shading Language (GLSL). The image is stored in a 2-D texture using a framebuffer object (FBO). The texture is then passed to a shader program to compute the pixel-wise absolute error between ref and pop. The sum is also performed on the GPU using the OpenCL implementation of the reduction operator in Boost.Compute [23, 8]. It provides the SAE in an effective manner.

Our implementation is based on a steady state evolutionary strategy (see Fig. 1). At each iteration of the optimisation process, a bad fly is selected for death and replaced with another one. We use a mutation operator to produce a new fly that is slightly different from the selected good fly. The aim is to create a new fly in the vicinity of a good fly. In this way the new fly is likely to be a good one too. Crossover is not used: If we consider two good flies located at the opposite corners of the image, a new fly in between is very likely to be bad.

The algorithm stops when a stopping criterion is met, e.g. maximum number of iterations, when the evolution process does not improve the performance of the whole population, or when the Threshold selection becomes too slow to find bad flies. A restart mechanism is eventually used to further improve the results. It allows the algorithm to leave a local minima (see Fig. 5).

5 Results

In this section we evaluate our method using the results obtained with five test images of increasing complexity (see Row 0 in Fig. 4). We consider the image reconstruction with 12 different schemes (see Table 1) using:

- Different image sizes: 256×256 or 512×512;
- Different colour quantisations: 60 colours or full RGB colours (i.e. 2^{24});
- Different types of tiles: square with a border, square without a border, set of lines, or flower;
- With or without restart mechanism.

For each test image, evolutionary reconstructions are obtained with these schemes:

- As a feasibility study, the first two images (Star and Yin & Yang) are relatively simple and have a relatively low resolution. 6 schemes of Table 1 are used.
- Three other test images are more complex. They are used to evaluate the 12 schemes.

Table 1: Summary of all the possible configurations used in Fig. 4.

#	256×256	512×512	60 colours	Full colour	One shader	Two shaders	Restart
1	✓			✓	✓		
2	✓		✓		✓		
3		✓		✓	✓		
4		✓	✓		✓		
5	✓			✓		✓	
6	✓		✓			✓	
7		✓		✓		✓	
8		✓	✓			✓	
9	✓			✓	✓	✓	✓
10	✓		✓		✓	✓	✓
11		✓		✓	✓	✓	✓
12		✓	✓		✓	✓	✓

Schemes 1 to 4 uses the algorithm presented in Fig. 1. The flies correspond to uniform rectangles (see "one shader" in Table 1). Schemes 5 to 8 uses the algorithm presented in Fig. 1 twice, with one restart between the two runs. The initial population of the first run is random (as in Fig. 3). The initial population of the second run is the best population of the first run. The flies correspond to complex shapes (see "two shaders" in Table 1). Examples of template shapes are given in Fig. 7. Different shader programs can be used to generate different effects depending on the pixel colour/intensity of the templates. Schemes 9 to 12 are almost similar to Schemes 5 to 8. The only difference is that the flies are uniform rectangles during the first run and complex shapes during the second run. The aim is to speed-up computations.

The algorithm is tested with two sets of parameters (see Table 2), one with a 22500-D search space, and the other one with a 45000-D search space. We fix the mutation probability to 100% because, as we saw previously, crossover is not suitable in our case. In practice, the only parameters chosen by the user are the image (and its size) and the number of individuals. The initial size of tiles is computed depending on the images size and the number of individuals (see Eq. 1). Note that the sizes will then encompass evolution. We empirically estimated suitable population sizes for different image resolutions. We balanced the number of flies and image size to avoid a premature convergence that would slow down the entire process.

Table 2: Parameters used to generate the images in Fig. 4.

Image size	256×256	512×512
Number of flies	2500	5000
Number of generations	40000	40000
Probability of mutation	100 %	100 %
Probability of crossover	0.0 %	0.0 %
Corresponding scheme	1, 2, 5, 6, 9 and 10	3, 4, 7, 8, 11 and 12
Number of unknowns	$9 \times 2,500 = 22,500$	$9 \times 5,000 = 45,000$

To assess which scheme from Table 1 is the most suitable one, Fig. 4 was printed on A3 paper and we individually asked 25 participants to indicate which image in each column they prefer. Table 3 shows the results in percentages. Strategies 11 and 12 are particularly popular among participants. To a lesser exten, the 10^{th} scheme is also popular, the 3^{rd} and 9^{th} also received some votes. Other schemes did not. Schemes 10, 11, and 12 use two shader programs and a restart mechanism. Scheme 11 uses full-colour in the original image, Scheme 12 did not.

Fig. 5 shows the evolution of global fitness with and without restart. It is well known that restart is useful in classical evolutionary algorithms where the solution of the optimisation problem is the best individual of the population. However, very little has been done for the cooperative co-evolution scheme of the Fly algorithm. In [26] a mitosis operator is used to double the size of the population. Here we keep the size of the population constant. When restart is not used, the evolution reaches a plateau then stagnates (see purple

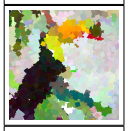
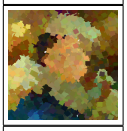
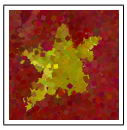
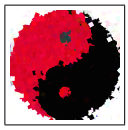
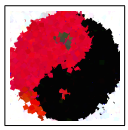
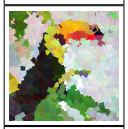
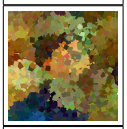
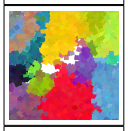
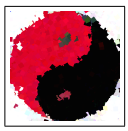
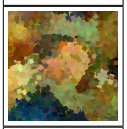
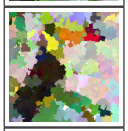
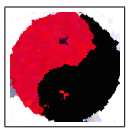
Scheme #	Star	Yin Yang	Glasses	Bird	Woman
0					
1					
2	N/A	N/A			
3					
4	N/A	N/A			
5					
6	N/A	N/A			
7					
8	N/A	N/A			
9					
10	N/A	N/A			
11					
12	N/A	N/A			

Figure 4: Evolutionary art using schemes of Table 1. The woman image (Fatima) is from the artist Lubna Ashrafis. Other test images are from the Open Images Dataset (https://github.com/openimages/dataset) under CC BY 4.0 license.

Table 3: Vote results (25 participants voted for their preferred image for each column in Fig. 4).

Scheme #	Star	Yin Yang	Glasses	Bird	Woman
3	0%	12%	20%	0%	0%
4	0%	0%	0%	0%	4%
9	12%	4%	8%	24%	4%
10	0%	0%	20%	**32%**	0%
11	**88%**	**84%**	**36%**	20%	**48%**
12	0%	0%	16%	24%	44%

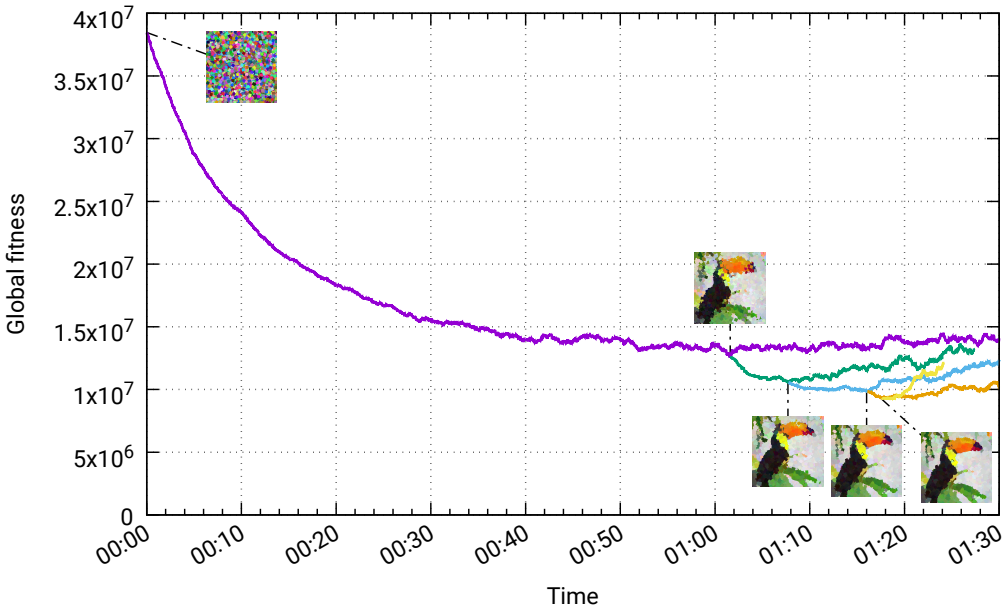


Figure 5: Evolution of the global fitness with 4 restarts. Images were computed using a Macbook Laptop with a 2.6 GHz Intel Core i5 CPU with an Intel Iris 5100 GPU.

curve). After the first restart, the global fitness decreases (see green curve). After the global minimum is found, it is possible that the global fitness increases. A similar phenomenon is observed for the subsequent restarts (see blue, yellow, and orange curves). This experiment demonstrates the benefit of a few restarts in the Fly algorithm.

During the generation of the images, tiles can be located at different depth to determine the colour of the closest (visible) tile. It is efficiently implemented using OpenGL's Z-buffer. Depending on the properties of the regions of the image, black tiles may be located behind red tiles, or *vice versa*. The algorithm picks up the discriminated edge between the black and red regions, no matter how small the regions are in the original image compared to the minimum size of a tile. For example, in Fig. 6b red tiles are over black tiles that are larger than the actual region in the original image. It also shows that our evolutionary algorithm chooses the right rotation angle to follow the curvature of the edges in the original image when colours are evolved. This is not as accurate when colours are picked up directly from the original image as in Fig. 6d. Edges are even blurrier in images generated using GIMP's filter (GIMPressionist) (see Fig. 6f).

In the following examples, four shapes (or masks) (see Fig. 7) to generate tiles: Square, flowers and stripes. Fig. 8 shows the results using the toucan as a test image. The shader program to generate the images can be altered to create different visual effects. The aim is to generate more appealing images. Figures 8a and 8b have been produced using the same mask (Fig. 7a) but with slightly different shader programs. Figures 8c and 8d have been generated using Fig. 7b as mask, but with a much bigger size and without considering the rotation angle of the tiles in the second image. The masks used in Figures 8e and 8f include an edge. More results are available as videos on YouTube at `http://tinyurl.com/ho5kfvb`.

To demonstrate the usefulness of our approach, further comparison examples have been done between the

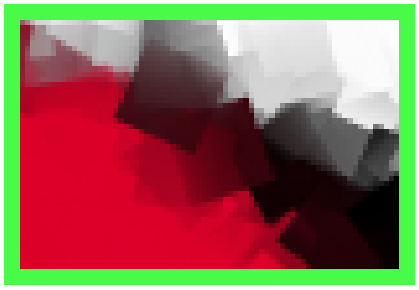
(a)

(b) ROI of 6a.

(c)

(d) ROI of 6c.

(e)

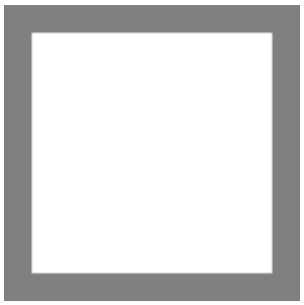


(f) ROI of 6e.

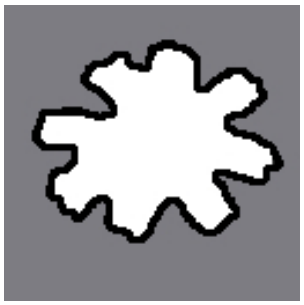
Figure 6: Edge and depth detection: 6a Image reconstructed using our method (evolving colours); 6c Image reconstructed using our method (without evolving colours); 6e Image reconstructed using GIMPressionist.

(a)

(b)

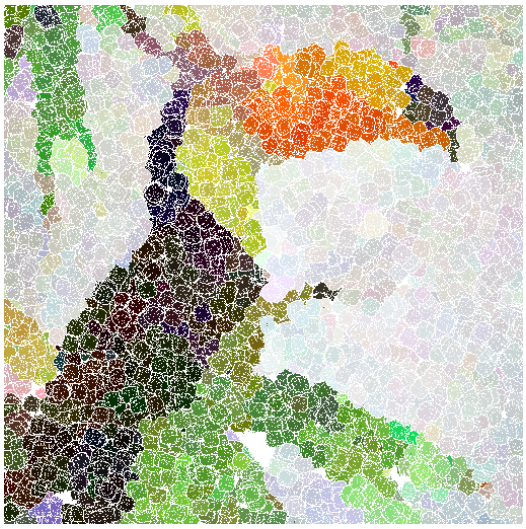


(c)



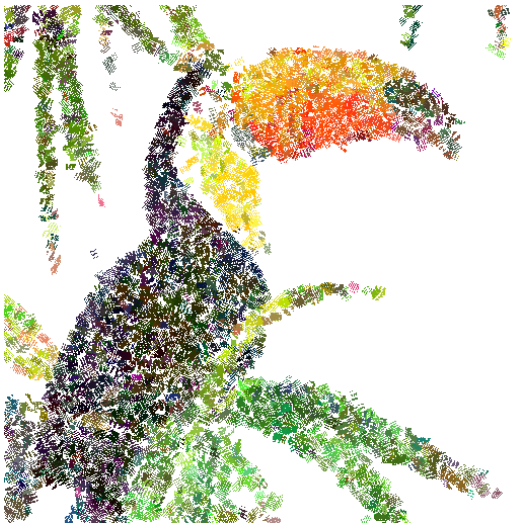
(d)

Figure 7: Examples of tile templates.

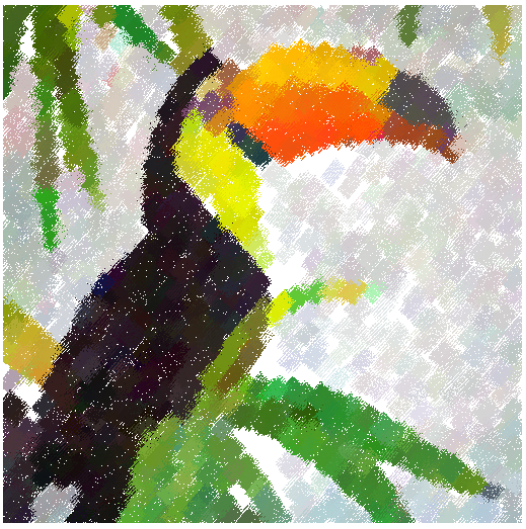


(a) Mask: Flower (Fig. 7a).

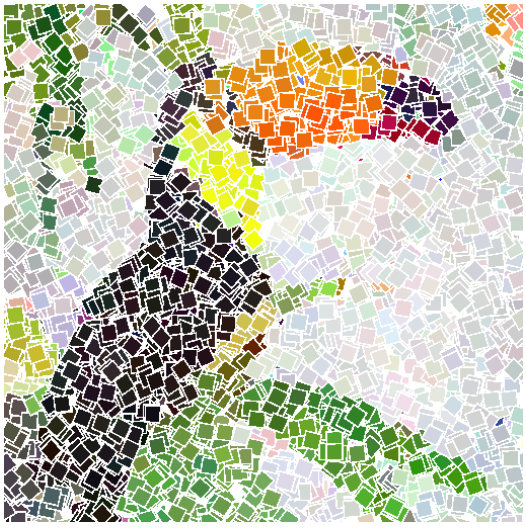
(b) Mask: Flower (Fig. 7a).



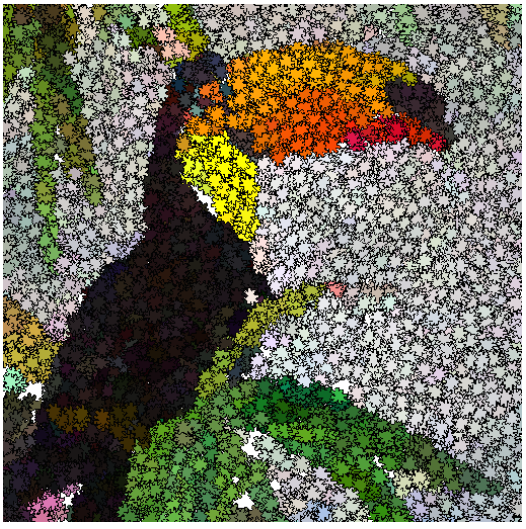
(c) Mask: Set of lines (Fig. 7b).



(d) Mask: Set of lines (Fig. 7b).



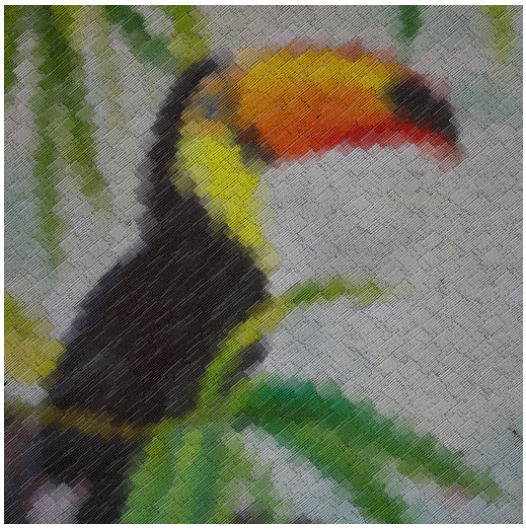
(e) Mask: square (Fig. 7c).



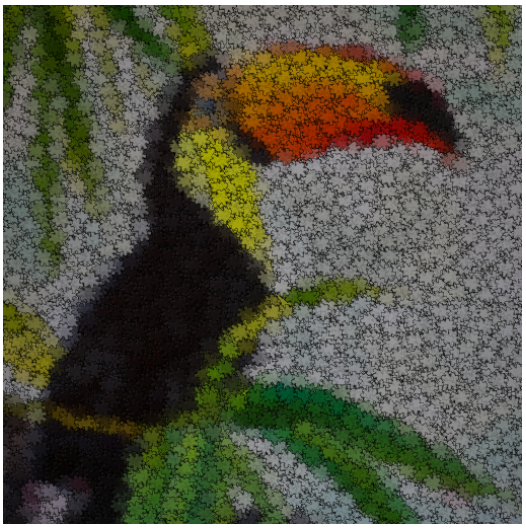
(f) Mask: square (Fig. 7d).

Figure 8: More appealing visual effects using different masks and shader programs.

proposed algorithm and GIMP using two types of mask: Flower and stripe (see Figures 8d, 8f and 9). Our method leads to better reconstructions in term of edges and colours (brightness).



(a) Mask: Set of lines (Fig. 7b).



(b) Mask: Flower (Fig. 7d).

Figure 9: Example of images produced with GIMP's filter (GIMPressionist).

6 Conclusion

The problem tackled here lies within the field of Evolutionary art. Our method relies on techniques inherited from CG, AE and scientific computing. We used an AE strategy based on the Fly algorithm for creating visual effects on an image in a fully-automatic fashion. The algorithm optimises the location of tiles in the 3-D space to approximate an input image. We used real-time CG rendering to generate the image data, and GPU computing to calculate the fitness functions. The algorithm can be modified to introduce multiple artistic visual effects. Different templates are used (square, flower, stripes) to define the shape of tiles.

Our initial proof-of-concept shows that the Fly algorithm can be used in Evolutionary Arts. Our implementation could be refined to take advantage of more advanced genetic operators, for example to speed-up computations, and to recover fine details. Further work will also include a more robust evaluation to ascertain that the resulting images are visually appealing. It will include a more extensive comparison study and an user evaluation survey. A friendly graphical user interface (GUI) will be added to introduce an optional level of user interaction. It will allow the user to control some parameters of the output image, e.g. shape of tiles, number of tiles, etc. A plugin for an image manipulation program, such as GIMP, will be released to make it available to potential users.

Acknowledgements

This work was partially supported by the European Commission, Grant no. 321968 (http://fly4pet.fpvidal.net), and the Iraqi Ministry of Higher Education and Scientific Research (MOHESR) .

References

[1] S. Battiato, G. D. Blasi, G. M. Farinella, and G. Gallo. Digital Mosaic Frameworks - An Overview. *Comput Graph Forum*, 26(4):794–812, 2007.

[2] A. M. Boumaza and J. Louchet. Dynamic flies: Using real-time Parisian evolution in robotics. In *Applications of Evolutionary Computing: EvoWorkshops 2001*, pages 288–297, 2001.

[3] N. S. H. Chu and C. L. Tai. Real-time painting with an expressive virtual Chinese brush. *IEEE Comput Graph*, 24(5):76–85, 2004.

[4] J. Collomosse. Evolutionary search for the artistic rendering of photographs. In J. Romero and P. Machado, editors, *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, pages 39–62. Springer, 2007.

[5] F. Devinck and L. Spillmann. The watercolor effect : Spacing constraints. *Vision Res*, 49(24):2911–2917, 2009.

[6] G. Elber and G. Wolberg. Rendering traditional mosaics. *Visual Comput*, 19(1):67–78, 2003.

[7] G. M. Faustino and L. H. De Figueiredo. Simple adaptive mosaic effects. In *Brazilian Symposium of Computer Graphic and Image Processing*, pages 315–322, 2005.

[8] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa. *Heterogeneous Computing with OpenCL*. Morgan Kaufmann, 1st edition, 2011.

[9] P. Haeberli. Paint by Numbers: Abstract Image Representations. *SIGGRAPH Comput. Graph.*, 24(4):207–214, Sept. 1990.

[10] A. Hausner. Simulating decorative mosaics. In *Proceedings of SIGGRAPH '01*, pages 573–580, 2001.

[11] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of SIGGRAPH '99*, pages 277–286, 1999.

[12] W. Hu, Z. Chen, H. Pan, Y. Yu, E. Grinspun, and W. Wang. Surface Mosaic Synthesis with Irregular Tiles. *IEEE T Vis Comput Gr*, 2626(c):1–13, 2015.

[13] T. Isenberg. *Visualization and Processing of Higher Order Descriptors for Multi-Valued Data*, chapter A Survey of Illustrative Visualization Techniques for Diffusion-Weighted MRI Tractography, pages 235–256. Springer, 2015.

[14] M. Kaplan and E. Cohen. Computer Generated Celtic Design. *Proceedings of the 14th Eurographics Workshop on Rendering 2003*, 44:9–19, 2003.

[15] R. M. Karp. *Complexity of Computer Computations*, chapter Reducibility among Combinatorial Problems, pages 85–103. Springer, 1972.

[16] B. Kaufmann, J. Louchet, and E. Lutton. Hand posture recognition using real-time artificial evolution. In *Applications of Evolutionary Computation: EvoApplicatons 2010*, pages 251–260, 2010.

[17] Y.-K. Lai, S.-M. Hu, and R. R. Martin. Surface mosaics. *Visual Comput*, 22(9):604–611, 2006.

[18] Z. Li, S. Qin, X. Jin, Z. Yu, and J. Lin. Skeleton-enhanced line drawings for 3D models. *Graph Models*, 76(6):620–632, 2014.

[19] J. Louchet. Stereo analysis using individual evolution strategy. In *Proceedings of the International Conference on Pattern Recognition*, volume 1 of *ICPR '00*, pages 908–911, 2000.

[20] J. Louchet, M. Guyon, M.-J. Lesot, and A. Boumaza. Dynamic flies: a new pattern recognition tool applied to stereo sequence processing. *Pattern Recogn Lett*, 23(1–3):335–345, 2002.

[21] J. Louchet and E. Sapin. Flies open a door to SLAM. In *Applications of Evolutionary Computation: EvoApplicatons 2009*, pages 385–394, 2010.

[22] L. Lu, F. Sun, H. Pan, and W. Wang. Global Optimization of Centroidal Voronoi Tessellation with Monte Carlo approach. *IEEE T Vis Comput Gr*, 18(11):1880–1890, 2012.

[23] K. Lutz. Boost.Compute. `http://boostorg.github.io/compute/`. Accessed: 2016-10-26.

[24] P. Mesejo, E. Fernández-blanco, A. B. Porto-pazos, E. F. Andez-blanco, and F. C. On. Artificial Neuron-Glia Networks Learning Approach Based on Cooperative Coevolution. *International Journal of Neural Systems*, 25(4), 2015.

[25] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, 2015.

[26] F. P. Vidal, J. Louchet, J.-M. Rocchisani, and É. Lutton. New genetic operators in the Fly algorithm: Application to medical PET image reconstruction. In *Applications of Evolutionary Computation: EvoApplicatons 2010*, pages 292–301, 2010.